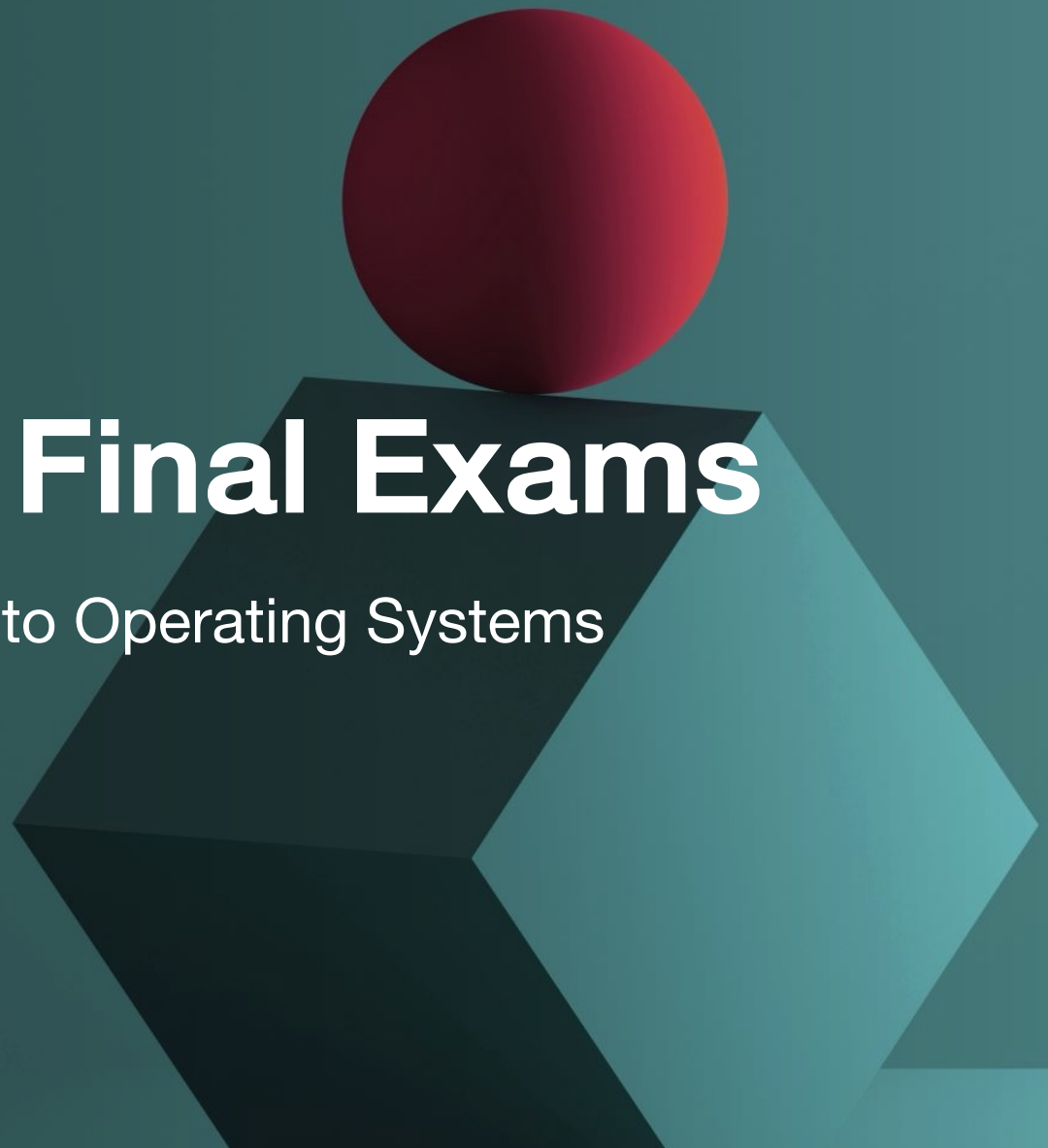


Past Year Final Exams

CS2106: Introduction to Operating Systems

AY23/24 Semester 1

Question 1

Consider a contiguous memory system with 1-byte allocation units. Which ONE of the following statements is FALSE?

A. Contiguous memory allocation is likelier to suffer from external fragmentation if it uses a best-fit allocation policy.

B. Contiguous memory allocation is likelier to suffer from internal fragmentation if it uses a worst-fit allocation policy.

C. In buddy allocation, we can test whether two blocks of size 2^n are buddies just by testing that bit n of the address of one block is a complement of bit n of the address of the other block, and that all higher order bits of the addresses are identical. We assume that the LSB of both addresses is bit 0.

D. It is possible to make worst-fit allocation run in O(1) by sorting the free memory blocks by size in descending order.

E. NONE of the statements A. to D. above are FALSE (i.e. ALL are TRUE)

Explanation

A contiguous memory allocation using 1-byte allocation units, strongly suggests that the system uses dynamic partitioning, hence it is unlikely to suffer from internal fragmentation.

Question 2

This question is tougher if you forgot the concept of direct mapped caching from CS2100

Which ONE of the following statements about disjoint memory schemes is TRUE?

A. All processes share a single system-wide page table.

B. We use multi-level page tables to increase speed of memory access.

C. The TLB is a direct-mapped cache.

D. In a system using inverted page tables, there is one inverted page table per process.

E. NONE of the above statements A. to D. are true (i.e. ALL are FALSE).

Explanation

A. Not true because each process has its own page table
B. Not true because with more levels of page tables, you need to access memory multiple times (accessing the page directory, then the actual page itself), and it would be worse if there are page faults.
C. Not true, TLB is a fully associative cache, where a frame in memory can be stored at any cache line in the TLB. (Direct mapped means a frame in memory is fixed to a specific cache line in the TLB, which is not the case)
D. If inverted page table is used, there will be only one inverted page table for the entire system.

Question 3

Which of the following statements i) to v) are TRUE? There may be multiple TRUE statements.

i. When integrating instruction/data caches with disjoint memory schemes, it is better to use virtual addresses to access the caches instead of physical addresses, so that we may be able to avoid doing an address translation.

ii. Increasing the size of the page table will generally reduce page fault rates.

iii. Locality causes an increase in page fault rates.

iv. A least recently used replacement policy can result in slow page replacements.

v. A large TLB will reduce the page-fault rate.

Explanation:

i. True. Think of the instruction and data cache as the TLB, since the entries in the TLB is the same as the page table, i.e. how to translate pages to frames, by using virtual address on the cache, you can potentially avoid address translation if the page entry exists in TLB.
ii. False. Belady's anomaly
iii. False. Spatial and temporal locality reduces page faults because the pages are already likely to be in memory
iv. True. It is computationally expensive to maintain the last access time of each page
v. False. Having a larger TLB does not help reduce the page-fault rate, it reduces the TLB miss rate.

Question 4

Which of the following statements i) to v) are TRUE? There may be multiple TRUE statements.

i. Two processes that share the same Open File Table (OFT) entry will still be able to read the file data without affecting each other in any way.

ii. You can have a maximum of 65536 blocks in a FAT16 file system

iii. Having larger logical blocks in an inode-based file system can cause external fragmentation.

iv. The minimum information required in a directory entry for a contiguous file system is the filename, starting block and length of the file in blocks.

v. It is possible so store file metadata in the FAT instead of in the directory.

Explanation:

i. False.

ii. False. Although theoretically speaking, there are 16 bits in the FAT entry to identify each block and $2^{16} = 65536$, but there are some block numbers that are reserved for EOF, Bad Sectors and FREE, so the actual number of usable blocks for pointing to the next block is less than 65536.

iii. False. Data Blocks pointed by each I-Node Data Pointer can be spread out across non-contiguous blocks in memory, hence it does not cause external fragmentation.

iv. True. Refer to the lecture slides

v. False. FAT does not store file metadata.

Question 5

A FAT32 file system is one where every FAT entry is 32 bits long. However, in the most common implementation only 28 bits are used for block numbers. In such an implementation, if the logical block size is 1024 bytes (1 KiB), what is the maximum disk/partition size possible with this file system, in GiB? (1 GiB = 1024 x 1024 KiB)?

A. 0.0625 GiB

B. 16 GiB

C. 256 GiB

D. 4096 GiB

E. None of the above options A. to D. are correct.

Explanation:

The maximum partition size possible is 2^{28} KB = $\frac{2^{28}KB}{1024 \times 1024KB} = \frac{2^{28}}{2^{20}}GB = 2^{8}GB = 256GB$

Question 6

2 bytes

We consider a system with 16-bit data words and instructions, and 32-bit byte addresses shown below. The memory space for a process consists of a text (i.e. code) segment, data segment, stack segment and heap segment. The segment registers store the segment IDs of the respective segment

Segment Register Values

Text Segment Register	2
Data Segment Register	1
Stack Segment Register	3
Heap Segment Register	4

Segment Table

Segment ID	Starting Address	Length
0	800	120
1	1200	800
2	3000	3125
3	8000	350
4	10000	403
5	15000	650
...	...	...

The “Start Address” column of the Segment Table contains the starting address of the segment as a 32-bit unsigned integer, while the length column shows the length of the segment in bytes as a 16-bit unsigned integer. The tables above show the segment register values for a particular process P. All addresses and lengths are shown in decimal.

Question 6

(a) In general (not just process P shown above), what is the maximum total size of a process in KiB? (1 KiB = 1024 bytes)

- A process has 4 segments, text, data, stack and heap.
- The length of each segment (in bytes) is represented using a 16-bit unsigned integer.
- Hence, the maximum size of each segment is 2^{16} bytes.
- So, the maximum total size of a process in KiB is $\frac{2^{16} \times 4}{1024} = \frac{2^{18}}{2^{10}} = 2^8 =$ 256 KiB

Question 6

(b) We are executing the following code fragment in this process P, and the segment numbers are as shown in the diagram above. All data accesses are from the Data Segment. Each instruction is 16 bits long and all instructions are fetched from the Text Segment. The starting logical address of array A is (1, 710).

```
        la $2, A              ; Load address of A into $2
        li $3, 0              ; Initialize $3 to 0
        li $5, 0              ; Initialize $5 to 0
Loop:   ld $4, [$2]           ; Load the word at address in $2 into $4
        add $5, $5, $4        ; Add $4 to $5
        addi $2, $2, 2        ; Increment $2 by 2
        addi $3, $3, 1        ; Increment $3 by 1
        blt $3, 50, Loop      ; Branch to Loop if $3 < 50
Exit: …                       ; Other irrelevant instructions.
```

Question 6

While knowing what each MIPS instruction does would be helpful for this question, the comments given at each line suffices as context for this question

i. What are the first 5 PHYSICAL addresses generated by the `ld` instruction? Fill your answers as decimal (base-10) integers.

- $2: Initially stores the address of array A, but gets incremented by 2 during each iteration of the loop
- Starting logical address for array A is (1, 710)
- Starting physical address for array A is 1200 (Base) + 710 (Offset) = 1910
  - Refer to the segment table provided for this question.
- First 5 physical addresses will be 1910, 1912, 1914, 1916 and 1918

Question 6

ii. What is the range of PHYSICAL addresses that the instructions shown are loaded from, assuming that the first instruction (la $2, A) is at (2, 1000)?

- Convert from logical address (2, 1000) to physical address 3000 + 1000 = 4000.
- Each instruction is 2 bytes and the system uses byte level addressing.
- So, the range of physical addresses are from 4000 to 4014
  - 4000 + (8 instructions – 1) * 2 bytes = 4014 [since first instruction is already at 4000]

```
        la $2, A                ; Load address of A into $2
        li $3, 0                ; Initialize $3 to 0
        li $5, 0                ; Initialize $5 to 0
Loop:   ld $4, [$2]             ; Load the word at address in $2 into $4
        add $5, $5, $4          ; Add $4 to $5
        addi $2, $2, 2          ; Increment $2 by 2
        addi $3, $3, 1          ; Increment $3 by 1
        blt $3, 50, Loop        ; Branch to Loop if $3 < 50
Exit: …                         ; Other irrelevant instructions.
```

Only 8 instructions were shown here

Question 6

iii. What is the value in $3 when this program suffers a memory violation error?

- Memory violation error happens when the physical address generated by the **`ld`** instruction exceeds 1200 + 800 – 1 = 1999.
  - Deduct 1 because 1200 is counted as the first address
- Since the starting address of the array is already at 1910 [from b(i)]
- It means that only $\frac{1999 - 1910}{2} + 1 = 45$ instructions can be loaded.
- Which means after incrementing 46 times, the physical address generated by the **`ld`** instruction becomes 2000 and causes the program to suffer from a memory violation error.
- Since $3 starts at value 0, after incrementing 46 times, the value of $3 will be 45.

Question 7

Segment Register Values

Text Segment Register	2
Data Segment Register	1
Stack Segment Register	3
Heap Segment Register	4

Segment Table

Segment ID	Starting Address	Length
0	800	120
1	1200	800
2	3000	3125
3	8000	350
4	10000	403
5	15000	650
…	…	…

Suppose we have the process P running the same program as in Question 6 with the same segment registers and segment table as before, but now the array A starts at logical address (1, 118). The first instruction of the program remains at (2, 1000).

Our system now has a disjoint memory system where each page/frame is 8 bytes.

```
        la $2, A              ; Load address of A into $2
        li $3, 0              ; Initialize $3 to 0
        li $5, 0              ; Initialize $5 to 0
Loop:   ld $4, [$2]           ; Load the word at address in $2 into $4
        add $5, $5, $4        ; Add $4 to $5
        addi $2, $2, 2        ; Increment $2 by 2
        addi $3, $3, 1        ; Increment $3 by 1
        blt $3, 50, Loop      ; Branch to Loop if $3 < 50
Exit: …                       ; Other irrelevant instructions.
```

Question 7

(a) How many pages are there in the text, data, stack and heap segments for process P?

Segment Register Values

Text Segment Register	2
Data Segment Register	1
Stack Segment Register	3
Heap Segment Register	4

Segment Table

Segment ID	Starting Address	Length
0	800	120
1	1200	800
2	3000	3125
3	8000	350
4	10000	403
5	15000	650
...	...	...

- Size of one page = 8 bytes
- Text Segment: [3125 / 8 bytes] = 391 pages
- Data Segment: 800 / 8 bytes = 100 pages
- Stack Segment: [350 / 8 bytes] = 44 pages
- Heap Segment: [403 / 8 bytes] = 51 pages

Question 7

(b) What is the total internal fragmentation for process P in bytes in this system?

- Text Segment: $\frac{3125}{8} = 390.625$
  - Internal Fragmentation = $0.375 \times 8 = 3$ bytes
- Data Segment: $\frac{800}{8} = 100$
  - Internal Fragmentation = 0 bytes
- Stack Segment: $\frac{350}{8} = 43.75$
  - Internal Fragmentation = $0.25 \times 8 = 2$ bytes
- Heap Segment: $\frac{403}{8} = 50.375$
  - Internal Fragmentation = $0.625 \times 8 = 5$ bytes
- Total Internal Fragmentation = 3 + 2 + 5 = 10 bytes

Segment Register Values

Text Segment Register	2
Data Segment Register	1
Stack Segment Register	3
Heap Segment Register	4

Segment Table

Segment ID	Starting Address	Length
0	800	120
1	1200	800
2	3000	3125
3	8000	350
4	10000	403
5	15000	650
...	...	...

Question 7

(c) What is the total external fragmentation in bytes possible in this system overall (not just Process P but considering all possible processes)?

0 Bytes (Since this system uses disjoint memory management)

(d) What are the first and last page numbers accessed by our program when reading array A?

- The starting logical address of Array A is (1, 118),
- Which makes the starting physical address of Array A = 1200 + 118 = 1318
- Since each page is 8 bytes, $\frac{1318}{8} = 164.75$ (Need 165 pages)
- Since page numbers start from 0, the first page number is 164
- 50 elements are read from Array A where each element is a data word of 16 bits = 2 bytes
- Hence, the offset between the first and last element in the array = $(50 - 1) \times 2 = 98$ bytes
- The last element's starting physical address 1318 + 98 = 1416
- Last Page Number = $\left\lfloor \frac{1416}{8} \right\rfloor = 177$

Question 7

(e) What are the page numbers of the first instruction (la $2, A) and last instruction (blt $3, 50, Loop)?

- From question 6(b) part (ii), we have calculated the range of physical addresses for the first till last instructions to be 4000 to 4014.
- Page number of first instruction = $\left\lfloor \frac{4000}{8} \right\rfloor = 500$ (Need 501 pages so page number is 500)
- Page number of last instruction = $\left\lfloor \frac{4014}{8} \right\rfloor = 501$ (Need 502 pages so page number is 501)

Question 8

We consider a virtual memory system with 512-byte pages/frames. The system has 4 byte words, a 20-bit virtual addressing space and a 16-bit physical addressing space. All addresses are byte addresses.

(a) What is the maximum number of pages and frames possible in this system?

Maximum number of pages $= \frac{2^{20}}{512} = \frac{2^{20}}{2^9} = 2^{11} = 2048$

Maximum number of frames $= \frac{2^{16}}{512} = \frac{2^{16}}{2^9} = 2^7 = 128$

Question 8

(b) Assume that we have a single-level page table (i.e. “direct paging”) with each entry containing a valid bit, a dirty bit and (of course) the frame number holding the page. How large is this page table, in KiB? One KiB is 1024 bytes. (Hint: The number of bytes in each page table entry must be a whole number).

- Since each page table entry includes the valid bit and dirty bit, two extra bits are needed.
- Since there can be a maximum of 128 frames, 7 bits are required to store the frame number.
- So, each page table entry requires $7 + 2 = 9$ bits = 1.125 bytes.
- But since the number of bytes in the page table entry must be a whole number, then we must round up 1.125 bytes to 2 bytes.
- So, the page table is $\frac{2048 \times 2}{2^{10}}$ KiB = 4 KiB

Question 8

(c) A particular process is allocated 3 frames, and accesses the following memory addresses in decimal:

1580, 2184, 1174, 1828, 3730, 2742, 1680, 3380, 2668, 2350

Assuming that we are using an LRU replacement policy. How many page faults are there given the memory accesses above?

- Since each page is 512 bytes, it would require 9 bits for offset.
- The next step is to convert each of the memory addresses to binary to get the page numbers.
- Finally, apply the LRU replacement policy on 3 frames, keep tracking of when each page was last accessed.

Question 8

Decimal	Binary	Page No.	Frame 1	Frame 2	Frame 3	Time	Page Fault?
1580	011 001000000	3	3 (0)			0	Y
2184	100 010001000	4	3 (0)	4 (1)		1	Y
1174	010 010010110	2	3 (0)	4 (1)	2 (2)	2	Y
1828	011 101000100	3	3 (3)	4 (1)	2 (2)	3	N
3730	111 010010010	7	3 (3)	7 (4)	2 (2)	4	Y
2742	101 010110110	5	3 (3)	4 (3)	5 (5)	5	Y
1680	011 010010000	3	3 (6)	4 (3)	5 (5)	6	N
3380	110 100110100	6	3 (6)	6 (7)	5 (5)	7	Y
2668	101 001101100	5	3 (6)	6 (7)	5 (8)	8	N
2350	100 100101110	4	4 (9)	6 (7)	5 (8)	9	Y

(c) There were 7 page faults

Question 8

(d) We now assume that this system has a TLB with a hit rate of 95%, a TLB access time of 3 ns, a main memory access time of 80ns, and that it takes 5 ms to remedy a page fault. Suppose that the page fault rate is 2% (Note: This is not necessarily the correct answer to the previous part). What is the average amount of time it takes to access memory? There is no other caching other than the TLB. The page table is assumed to be fully in memory.

0.95 * (3ns + 80ns) + 0.05 * (0.98 * (3ns + 80ns + 80ns) + 0.02 * (3ns + 80ns + 5ms))
= 0.95 * (3ns + 80ns) + 0.05 * (0.98 * (3ns + 80ns + 80ns) + 0.02 * (3ns + 80ns + 5000000ns))
= 5086.92ns

Question 8

(e) Let's now suppose that we use a 2-level page table where there are 256 second-level page tables. What is the minimum amount of memory in bytes required to hold the page tables in order to be able to access a memory location?

- Originally, there were 2048 pages. If there are 256 second level page tables, then each second level page table contains 8 pages
- Each second level page table requires $8 \times 2\ bytes = 16\ bytes$
- The minimum amount of memory would amount to one entry in the page directory pointing to one second level page table consisting of 8 pages while the other page directory entries remain uninitialized.
- Since the question wasn't very clear what was in the page directory entry, two answers were accepted.
- Answer 1:
  - To uniquely identify 256 directory entries, we need 8 bits or 1 byte for each entry.
  - Hence the minimum amount of memory required to hold the page tables is 256 + 16 = 272 bytes
- Answer 2:
  - Each directory entry will store the full 20-bit virtual address to each second-level page table.
  - Hence the minimum amount of memory required is $\frac{256 \times 20}{8} + 16 = 656\ bytes$

Question 9

Let's consider a contiguous memory allocation scheme using buddy allocation, with 4-byte allocation units. This works the same way as what you've seen in the lectures, except that entry n in the table now points to blocks of 2^n allocation units rather than 2^n bytes.

Suppose we have a total of 1024 bytes of memory starting at memory address 0, and the following requests are made:

1. Allocate 45 bytes of memory
2. Allocate 300 bytes of memory.
3. Allocate 65 bytes of memory.
4. Deallocate memory requested in 1.
5. Allocate 15 bytes of memory
6. Deallocate memory requested in 2.
7. Allocate 125 bytes of memory.

Question 9

(a) How many allocation units in total do we have in this system?

1024 bytes / 4 bytes = 256 allocation units

(b) What are the addresses of the memory locations requested in steps 3, 5 and 7? Remember that we are now allocating in units of 4 bytes and not 1 byte. Also, if multiple blocks of the correct size are available, we always allocate from the block with the lowest starting address.

45/64				65/128		300/512
				65/128		300/512
15				65/128		300/512
15				65/128		
15				65/128	125/128	

Question 9

(b) What are the addresses of the memory locations requested in steps 3, 5 and 7? Remember that we are now allocating in units of 4 bytes and not 1 byte. Also, if multiple blocks of the correct size are available, we always allocate from the block with the lowest starting address.

- Step 3: 128 bytes ÷ 4 bytes = 32
- Step 5: 0 ÷ 4 bytes = 0
- Step 7: 256 ÷ 4 bytes = 64

(c) What is the total internal fragmentation at the end of step 7 above?

(16 - 15) + (128 - 65) + (128 - 125) = 1 + 63 + 3 = 67 bytes

Question 10

Consider the program below:

```
uint16_t data; // Data is an unsigned 16 bit integer
int fd = open("test.txt", O_RDONLY);
for(int i=0; i<1048576; i++)
      read(fd, &data, sizeof(data));
```

Suppose each disk block holds 512 bytes of data. Block pointers are 16-bit unsigned integers.

Question 10

(a) In the worst case, what is the total number of disk blocks read on a FAT-16 system, assuming that the FAT is already in memory?

- In each iteration of the for loop, 16 bits of data is read from “test.txt” equivalent to 2 bytes
- Exiting the for loop, $1048576 \times 2 = 2097152$ bytes were read.
- Since, each disk block holds 512 bytes of data, $\frac{2097152}{512} = 4096$ blocks of data were read.

(b) In the worst case, what is the total number of disk blocks read on a linked-list system? Assume that the catalog entry for the file contains both the first and last block numbers of the file.

The linked list system has no FAT, since the pointer is in each block, each block can only store 512 – 2 = 510 Bytes.

So in total, $\left\lceil \frac{2097152}{510} \right\rceil = 4113$ blocks are read.

Question 10

Clarified with Prof Colin, that index blocks are NOT in memory even through the inode for a file is already in memory.

(c) Suppose we have an inode system with 4 direct pointers, 2 single-level indirect pointers and 1 double-level indirect pointer:

i. What is the largest possible file size in MiB with this inode structure? 1MiB = 1024 KiB, 1 KiB = 1024 bytes. Calculate to 2 decimal places.

- Since block pointers are 16-bit unsigned integers, then each block pointer is 2 bytes.
- So, the maximum number of block pointers in a directory block is $\frac{512}{2} = 256$
- The number of blocks pointed by each single-level indirect pointer is 256 blocks
- The number of blocks pointed by each double-level indirect pointer is 256^2
- In total, there can be a maximum of $256^2 + 256 + 256 + 4 = 66052$ blocks in this inode structure
- Hence, the largest possible file size in MiB is $\frac{66052 \times 512}{1024 \times 1024} =$ 32.25 MiB

ii. Given our program above, how many blocks in total are read, assuming that the inode is already in memory?

- Since we need to read 4096 blocks in total, the first 4 blocks can be directly read
- The 2 single-indirect pointers gives us 512 blocks, but we also need to read 2 extra directory blocks
- For the double level indirect pointer, we will read 1 index block, then $\left\lceil \frac{4096 - 4 - 512}{256} \right\rceil = 14$ second level index blocks are read.
- In total, there will be 17 index blocks (14 + 1 + 2)
- In total, there are 4096 + 17 = 4113 blocks are read.

AY23/24 Semester 2

Part 2: Question 4

We consider an inode file system with 8 direct pointers, 4 single-indirect pointers, 2 double-indirect pointers, and 1 triple-indirect pointer. Each data block is 1024 bytes, and each block pointer is 8 bytes.

a. (2 marks) What is the maximum file size possible on this file system? Express your answer in GB (1 GB = 1024 MB = 1024 x 1024 KB = 1024 x 1024 x 1024 bytes) correct to 2 decimal places. Assume that there is sufficient space available.

- Each data block can store $\frac{1024}{8} = 128$ pointers
- Total Blocks in I-Node = $8 + 4 \times 128 + 2 \times 128^2 + 128^3 = 2130440$
- Maximum File Size = $\frac{2130440 \times 1024}{1024 \times 1024 \times 1024} = 2.03$ GB

Part 2: Question 4

b. (8 marks) Assuming that only the inode has been loaded into memory, and none of the lower level index blocks have been loaded yet, what is the TOTAL number of blocks that must be loaded from disk when reading data from the following byte offsets (relative to the start of the file):

Offset (bytes from start)	Total # of Blocks Read
67	
8100	
53450	
34082815	

Part 2: Question 4

Indirection Level	Block No.	N-th Block
0	0 to 7	1 to 8
1	8 to 519	9 to 520
2	520 to 33287	521 to 33288
3	33288 to 2130439	33289 to 2130440

b. (8 marks) Assuming that only the inode has been loaded into memory, and none of the lower level index blocks have been loaded yet, what is the TOTAL number of blocks that must be loaded from disk when reading data from the following byte offsets (relative to the start of the file):

- Step 1: Get the block numbers by doing $\left\lfloor \frac{offset}{1024} \right\rfloor$
- Step 2: Determine the indirection level of the block
- Step 3: Need to account for loading directory block of pointers when calculating the number of blocks

Offset (bytes from start)	Block No.	N-th Block	Indirection Level	Total # of Blocks Read
67	$\left\lfloor \frac{67}{1024} \right\rfloor = 0$	1	0	1
8100	$\left\lfloor \frac{8100}{1024} \right\rfloor = 7$	8	0	1
534550	$\left\lfloor \frac{534550}{1024} \right\rfloor = 522$	523	2	1 + 1 + 1 = 3
34082815	$\left\lfloor \frac{34082815}{1024} \right\rfloor = 33283$	33284	2	1 + 1 + 1 = 3

AY24/25 Semester 1

Note

- The question number on Examplify is given by Question #: <No.>.
- There is another question number (e.g. Q1b) which is the question number which is used by examiners and invigilators for clarifications.
- The question numbers used in this slides will follow the question number used in **Examplify**.

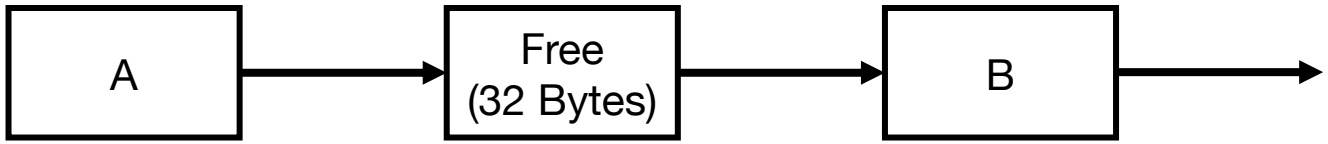
Context for Q1 to Q2

We consider a system with 256 bytes of memory and 8-byte allocation units (i.e. memory is always allocated in units of 8 bytes). All addresses are byte addresses. The current state of the memory is shown below. If a partition is allocated to a request, the request name (a single capital letter, e.g. A, B, C etc) is shown. Otherwise, the partition is shown as “Free”.

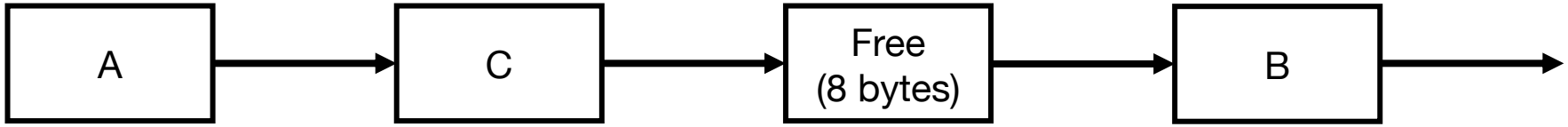
Start Request	End Request	Status
0	63	A
64	79	Free
80	103	B
104	127	Free
128	191	C
192	223	Free
224	231	D
232	255	Free

Context for Q1 to Q2

We use the next-fit allocation algorithm to allocate memory. In this algorithm, we start the search for the first free partition from the beginning of the free list. For subsequent requests, we continue from where we stopped. If we performed an allocation, we resume our search at the very next free partition from where we complete our allocation. For example, suppose we have the following memory state:



Suppose we allocate 24 bytes in the free partition above to C, we get:



Context for Q1 to Q2

The search for the next free partition will start at the 8-byte free space shown above. Note that an operation to free a partition does not make that partition the next one to be searched.

We have the following five requests:

1. E: Allocate 20 bytes
2. F: Allocate 12 bytes
3. Free memory allocated to D
4. G: Allocate 30 bytes
5. H: Allocate 12 byte

Question 1

We have the following five requests:

1. E: Allocate 20 bytes
2. F: Allocate 12 bytes
3. Free memory allocated to D
4. G: Allocate 30 bytes
5. H: Allocate 12 byte

What are the starting addresses of E, F, G and H? (10 marks)

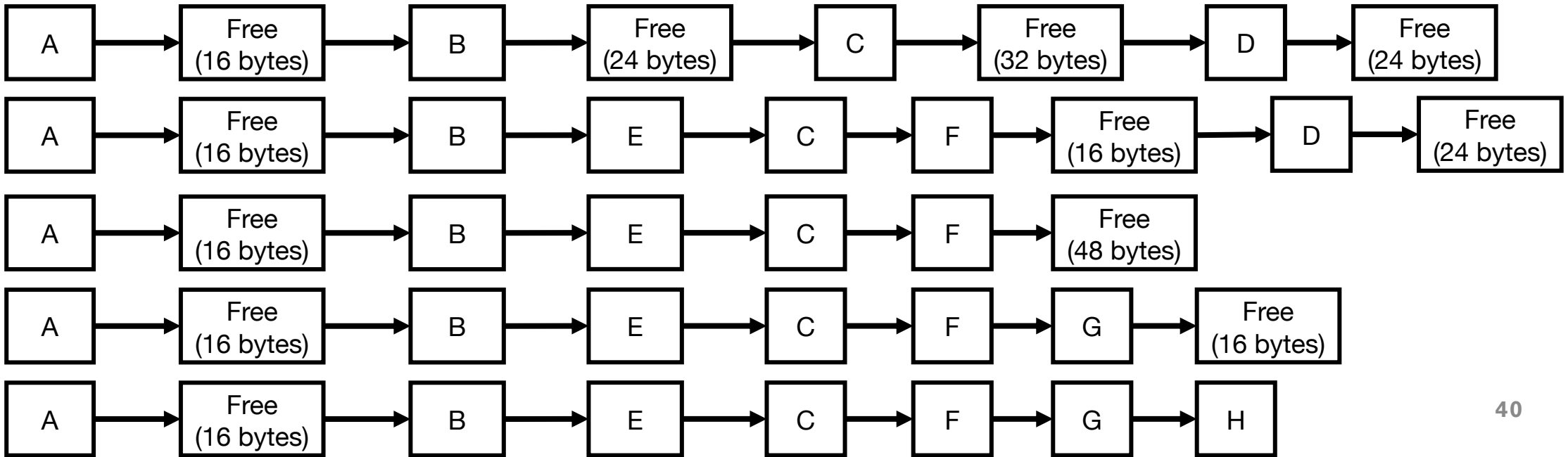
E: 104

F: 192

G: 208

H: 240

Tip: Round up the amount of memory allocated in each request to the nearest unit of 8 bytes.
E (24 bytes), F (16 bytes), G: (32 bytes), H (16 bytes)



Question 2

i. What is the total internal fragmentation after completing all of the requests above? Do not include memory allocated to A, B and C since we don't know what the original requests were. (1 mark)

Total Internal Fragmentation = (24 – 20) + (16 – 12) + (32 – 30) + (16 – 12) = 14 bytes

i. How much free memory is left after all the requests have been fulfilled? (1 mark)

16 bytes

Context for Q3 and Q4

We now consider a system again with 256 bytes of memory, but this time with an allocation unit of 1 byte (i.e. we can allocate blocks as small as 1 byte), and buddy allocation. We always allocate from the lowest address block first. So, if we have two buddy blocks of 64 bytes at address 0 and 64, we allocate the block at address 0 first.

The memory is currently in the state shown below. As before, a partition allocated to a request is indicated by a single alphabet, while free partitions are indicated by the word “Free”. To help you, the binary encoding of the starting addresses is shown.

Start Address	End Address	Status
0b0000 0000 (0)	15	Free
0b0001 0000 (16)	31	A
0b0010 0000 (32)	63	Free
0b0100 0000 (64)	127	B
0b1000 0000 (128)	159	C
0b1010 0000 (160)	255	Free

Question 3

We have the following four requests:

1. E: Allocate 10 bytes
2. F: Allocate 92 bytes
3. G: Allocate 28 bytes
4. Free(B)

Fill in the table below showing the state of the memory after fulfilling all of the requests above. Two or more adjacent blocks of free memory must be shown as a single contiguous block; for example, if you have free memory from addresses 64 to 95 and 96 to 127, be shown as a single contiguous block from 64 to 127, or it will be marked as incorrect.

Memory State. In status fill a letter (e,g, A, B, G, etc) of the request if the partition is allocated to a request or “Free” without the quotes if the partition is free in the table shown on Examplify: (Note: due to a limitation in Examplify, only the first five rows of the memory state table are shown):

Question 3

Start Address	End Address	Status
0	15	E
16	31	A
32	63	G
64	127	Free
128	159	C

(8 marks)

Question 4

What is the total internal fragmentation after all the requests in Q3 are fulfilled? Do not count memory already in use before the requests were made.

(16 - 10) + (32 – 28) = 6 + 4 = 10 bytes

Context for Q5 to Q8

We consider a system with a 32-bit virtual address space and a 24-bit physical address space. Pages/frames are 16 KiB (1 KiB = 2^{10} bytes) long. Every page table entry (PTE) consist of a frame number, three access bits (wrx), an “in-memory bit”, and a dirty bit.

Question 5

How large is each page table entry, rounded up to the nearest byte?

(2 marks)

- Given a 24-bit physical address space, we can have up to 2^{24} addresses
- Number of frames = $\frac{2^{24}}{16 \times 2^{10}} = 2^{10} = 1024$ (Need 10 bits for frame number)
- Total number of bits required in a PTE
  - 10 (Page Number) + 3 (Three Access Bits) + 1 (In-Memory Bit) + 1 (Dirty Bit) = 15 Bits
  - Rounded to the nearest byte = $\left\lceil \frac{15}{8} \right\rceil = 2$ bytes

Question 6

If we had every entry of the page table in memory, how large would this page table be? State your answer in KiB (1 KiB =2^{10} bytes)

(3 marks)

- Number of pages = $\frac{\text{Number of virtual addresses}}{\text{Page Size}} = \frac{2^{32}}{16 \times 2^{10}} = 2^{18}$
- Page Table Size = Number of pages × Page Size = $\frac{2^{18} \times 2}{2^{10}} = 512\ KiB$

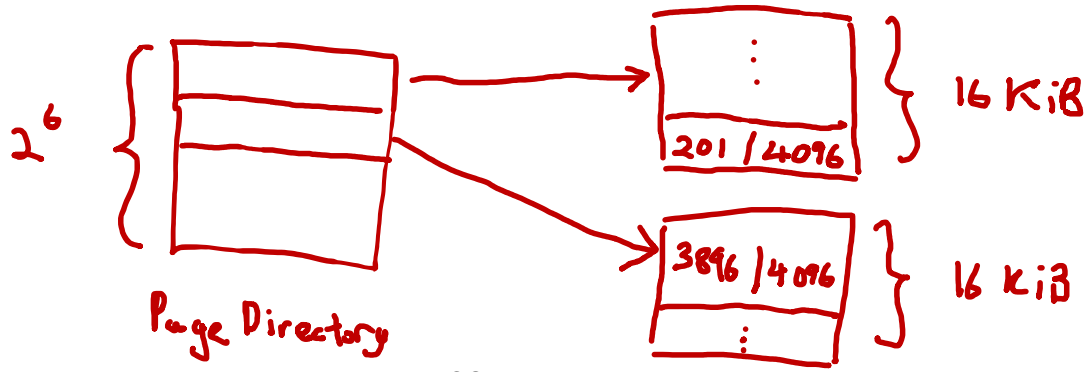
Question 7

If we used multilevel page tables, what is the maximum number of levels that we would have if PTEs are the same size at every level, each PTE is 4 bytes long, and we fully make use of a page to store PTEs? The page table directory counts as one level, although it may not occupy an entire page and may have fewer entries than the lower level page tables.

(4 marks)

- Number of PTEs at every level = $\frac{\text{Page Size}}{PTE\ Size} = \frac{16 \times 2^{10}}{4} = 2^{12}$
  - Need 14 bits for byte offset in a 16 KiB page
  - Need 12 bits for each level
- Calculations
  - 32 – 14 = 18 (Bits used for multi-level paging)
  - Number of levels = $\left\lceil \frac{18}{12} \right\rceil = 2$
    - 12 bits for page table
    - 6 bits for page table directory

Question 8



We have an array of size 64 MiB (1 MiB = 2^{20} bytes) starting at address 0x2FCDFB3C. How much memory is used by your page tables, including the page directory, in our multilevel page table? Express your answer in bytes.

(3 marks)

- Convert 0x2FCDFB3C to Binary
  - 0010 1111 1100 1101 1111 1011 0011 1100
  - Byte Offset = 11 1011 0011 1100 (Since Byte Offset is not 0, 1 extra page is needed)
  - Page No = 1111 0011 0111 (Since Page No is not 0, 1 extra directory entry needed) → 3895
  - Page Table No = 0010 11
- Calculate number of pages need for 64 MiB
  - $\frac{64 \times 2^{20}}{16 \times 2^{10}} = \frac{2^{26}}{2^{14}} = 2^{12}$
  - Need 2 second-level page tables: $2 \times 16\ KiB = 32768\ Bytes$
  - Need 1 page directory: $2^6 \times 4 = 256\ Bytes$
- Memory used = $32768 + 256 = 33024\ Bytes$

Context for Q9 and Q10

Consider a system with a single-level page table, and with the following characteristics:

TLB Hit Rate	96%
TLB Access Time	2ns
Memory Access Time (Read)	50ns
Cache Memory Hit	97%
Cache Access Time	1ns
Page Fault Rate	2%
Disk Access Time	25 ms

Note: 1 ns = 10^{-9} s, 1 us = 10^{-6} s, 1 ms = 10^{-3} s

Context for Q9 and Q10

Some important points:

1. Some pages are dirty and may need to be written back to disk before being replaced.
2. The time taken to see if a page table entry is in TLB is 2 ns. If the page table entry is in TLB, the time taken to get the frame number is negligible.
3. In the event of a TLB miss, the TLB is not re-read after remedying the miss.
4. The time taken to check the cache for a hit is 1 ns. If there is a cache hit, the time taken to read/write the cache block is negligible.
5. In the event of a cache miss, the cache needs to be re-read after remedying the cache miss. We ignore the time taken to write the memory block to cache.
6. The page table entries are only cached in the TLB and not in the memory cache.
7. In the event of a page fault, the faulting memory location must be re-accessed after remedying the page fault.
8. For any answer that is not an integer, provide your answer to three decimal places.
9. Ensure that you provide your answers in the units shown.

Question 9

(a) What is the worst-case access time in this memory hierarchy? State your answer in ns. (1 mark)

- TLB Miss = 2ns (Check TLB) + 50ns (Access Page Table in Memory) = 52ns
- Remedy Page Fault = 25ms (Write Dirty Page back to Disk) + 25ms (Retrieve Requested Page from Disk) + 50ns (Access Memory) = 50ms + 50ns
- Worst Case Access Time = 52ns + 50ms + 50ns = 50000102ns
- Note: Once it is known that the requested page is not resident in memory after accessing the page table, there is NO point for the OS to access the cache.

(b) What is the best-case access time in this memory hierarchy? State your answer in ns. (1 mark)

2ns (TLB Hit) + 1ns (Cache Memory Hit) = 3ns

Question 10

What is the average memory access time in this memory hierarchy, assuming that 25% of the pages chosen for replacement are dirty and must be written back? State your answer in ns. (6 marks)

- Average Time to Read Memory with Cache
  - = 0.97 × 1ns + 0.03 × (time to read cache + time to read memory + time to reread cache)
  - = 0.97 × 1 + 0.03 × (1 + 50 + 1) = 2.53ns
- Cases to Consider
  - TLB Hit = 2ns (TLB) + 2.53ns = 4.53ns
  - TLB Miss + Page in Memory = 2ns (TLB) + 50ns (Access Page Table) + 2.53ns = 54.53ns
  - TLB Miss + Page in Secondary Storge = 2ns (TLB) + 50ns (Access Page Table) + 25ms (Disk Access) + 0.25 × 25ms (25% written back) + 50ns (Memory Access) = 31250102ns
- Average Memory Access Time:
  - $0.96 \times 4.53 + 0.04 \times (0.98 \times 54.53 + 0.02 \times 31250102) = 25006.57$ns

Question 11

Consider an inode-based file system with the following characteristics:

- Size of each block: 4 KiB (1 KiB = 1024 bytes)
- Size of each block pointer: 4 bytes
- Number of direct pointers: 12
- Number of single indirect pointers: 2
- Number of double indirect pointers: 1
- Number of triple indirect pointers: 1
- Average time taken to read a block: 12 ms (1 ms = 10^{-3} s)

For any answer that is not an integer, give your answer to three decimal places.

Question 11

(a) What is the maximum number of blocks possible on this partition? State your answer to two decimal places in billions of blocks (1 billion = 10^9) (2 marks)

- A partition contains multiple I-Nodes.
- Each Block Pointer is 4 bytes or 32 bits.
- Maximum number of blocks = $2^{32} = 4294967296 = 4.29$ billion blocks

(b) What is the maximum file size possible in this file system? Express your answer to two decimal places in GiB (1 GiB = 2^{30} bytes) (2 marks)

- Number of pointers in index block = $\frac{4 \times 1024}{4} = \frac{4096}{4} = 1024$
- Maximum number of blocks = $12 + 2 \times 1024 + 1024^2 + 1024^3 = 1074792460 = 1.07$ billion blocks
- Maximum File Size = $\frac{1074792460 \times 4096}{2^{30}} = 4100.01$ GB

Question 11

(c) What is the best-case average time for reading data from a file? Express your answers in milliseconds (1 ms = 10^{-3} seconds) (2 marks)

- Best Case Average Time = Reading a block from a direct pointer = 12 ms

(d) What is the worst-case average time for reading data from a file? Express your answers in milliseconds (1 ms = 10^{-3} seconds)

- Worst Case Average Time = Reading a data block in the triple indirect pointer
- Need to read 3 index blocks and 1 data block
- Worst Case Average Time = $4 \times 12 = 48$ms